International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 8, No. 2, July 2019, pp. 99~106
ISSN: 2089-4864, DOI: 10.11591/ijres.v8.i2.pp99-106

Low power and high performance FFT with different radices

Md. Zakir Hussain[1], Kazi Nikhat Parvin[2]
[1]Department of Electronics and Communication Engineering, Muffakham Jah college of Engineering and Technology, Hyderabad, India
[2]Department of Electronics and Communication Engineering, Bhoj Reddy Engineering College for Women Hyderabad, India

Article Info

Article history:

Received Jan 25, 2019
Revised Apr 3, 2019
Accepted May 15, 2019

Keywords:

Fast Fourier Transform (FFT) computational elements
Radix-2
Radix-2^2
Radix-2^3 FFT architecture
Radix-4

ABSTRACT

FFT is one of the most active blocks in digital signal processing and in various field of communication systems. FFT has received significant attention over the past years to increase its capability and versatility. This paper describes an extensive study on trade-off of different radices with different computational elements of butterfly such as adders and multipliers. Finding an efficient radix along with computational elements is the key point to find best suite i.e. high precision, low power and low area applications like radar, filtering, image compression etc. The work also considers the precision and the data format to represent constant value such as Q-point. The proposed FFT architectures not only uphold better solutions for low power and high-performance application systems, but also open up a new research lines. This paper demonstrates that radix-2^3 consumes 43% less LUTs and 17% less power consumption, 40% increase of frequency in radix-2^2 in comparison with radix-2 algorithm for the combination of CSA with modified booth multiplier and the increment of frequency about 19%, 26% less LUTs consumption and 26% less power in Radix-2^2 when compared to radix-4 with various combination of adder and multiplier. In this work we have used Xilinx 14.7 XST for synthesis and the target device used is Spartan6 XC6SLX100. Simulation is carried out in Xilinx ISIM and also performed timing analysis and generated post-place and route.

Corresponding Author:

Md. Zakir Hussain,
Department of Electronics and Communication Engineering,
Muffakham Jah college of Engineering and Technology,
Hyderabad, Telangana-500005, India.
Email: zakir.hussain@mjcollege.ac.in

1. INTRODUCTION

Fast Fourier transform (FFT) was developed by Cooley-Tukey in 1965 and it plays a vital role in many arenas. FFT is an efficient class of computational algorithms to speed up DFT as FFT requires less computations due to its process of recursion known as butterfly [1]. Today's communication market features for strong competition regarding news standards. Fourier transform converts time domain to frequency domain and vice versa, FFT rapidly prefers such transformations. The radix 2 algorithm is well known simple algorithm for FFT processors, but it requires many complex multipliers [2]. As we move on to higher radices the number of twiddle factor decreases. FFT requires more computational elements while computing butterfly units in the radices. As multiplication utilizes large area and consumes more power when implemented on hardware. The complex computational elements should be reduced i.e. the complex multipliers and adders to make efficient FFT processor. The adders are very simple and easy to compute when compared to multipliers. Multiplier is required while multiplying the input with the twiddle factor in every radix [3].

This paper is briefly discussed about the different 16-point radices (radix-2, radix-4, radix-2² and radix-2^3) FFT algorithms using different computational elements (multipliers and adders) and to know the overall impact on FFT processor. Fixed point representation is implemented while multiplying the input with the twiddle factor. Trivial and non- trivial multiplication is existing in the radix-4 FFT algorithms [4, 5]. Usage of high radix with higher bit width gives the high precision value in fixed point representation and used in radar applications, encoding the image. The remaining sections in the paper is organized as follows. In section II, the paper describes about the radix-2 algorithm. Section III discussed about the radix-4 algorithm. Section IV explains about the radix-2i algorithms (including radix-2^2 and radix-2^3). Section V clearly gives the idea regarding the proposed work. Section VI represents the synthesis result of different radices and section VII includes conclusion.

2. RADIX-2 ALGORITHM

Radix-2 FFT algorithm simple radix in used in FFT. The original input vector, x(n) is divided into two N/2 length vectors i.e. even and odd input terms (xe(n), xo(n)) [6, 7]. The equation is defined as,

$$X(K)=\sum_{N=0}^{N-1} \boldsymbol{x(n)W_n^k} \quad (1)$$

$$X_{odd}(n) = X(2n)$$

$$X_{even}(n)= X(2n+1)\ n= 0, 1\ N/2-1$$

The radix-2 DIT FFT is rewritten by deriving the equation

$$\boldsymbol{X(k) = \sum_{m=0}^{\frac{N}{2}-1} x(2m) w_N^{2mk} \sum_{m=0}^{\frac{N}{2}-1} x(2m+1) w_N^{(2m+1)k}} \quad (2)$$

The above equation divides radix-2 in even index inputs and the odd index inputs and then combines the two results to produce the entire DFT sequence [8, 9]. From the figure, it is observed that the second input gets multiplied with the twiddle factor and added with the first input to get the first output. Similarly, the second output is obtained by subtracting the multiplied term with the first input. Figure 1 shows signal flow graph of 16 Point radix-2 DIT-FFT.

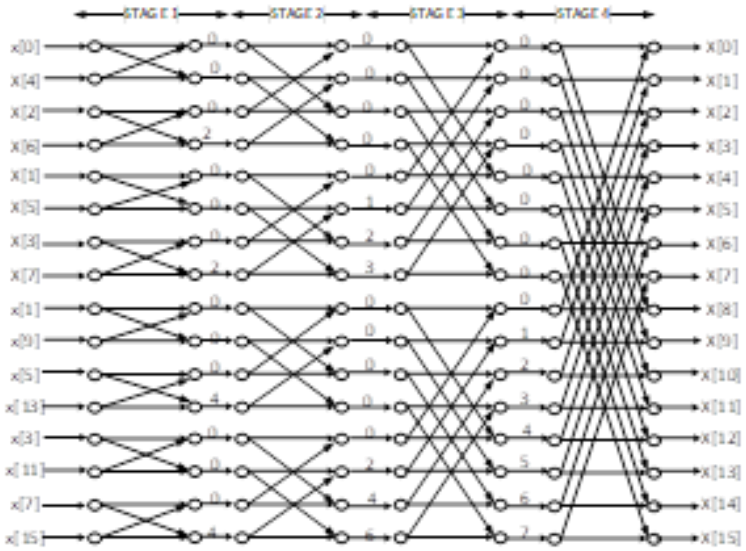

Figure 1. Signal flow graph of 16 point radix-2 DIT-FFT

3. RADIX-4 FFT ALGORITHM

The radix-4 FFT algorithm consists of two stages when compared to radix-2 FFT. In radix-4 algorithm the point size increases as the multiple of four. Hence, the radix-4 FFT requires fewer stages and butterflies than the radix-2 algorithm. The main idea of radix-4 DIT FFT is to divide the original input sequence into four smaller subsequences [10]. Figure 2 shows signal flow graph of radix-4 FFT for N=4. Figure 3 shows signal flow graph of 16 point radix-4 DIF-FFT.

X(K)=∑_(N=0)^(N-1)▒〖x(n)W_N^nk 〗 , where k= 0, 1, ….N-1

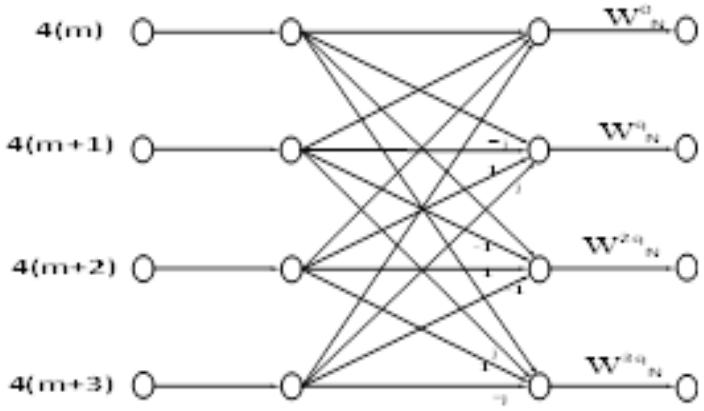

Figure 2. Signal flow graph of radix-4 FFT for N=4

The radix-4 algorithm equation is derived by rewriting (1)

$$X(k) = \sum_{m=0}^{\frac{N}{4}-1} x(4m)W_N^{k(4m)} + \sum_{m=0}^{\frac{N}{4}-1} x(4m+1)W_N^{k(4m+1)} + \sum_{m=0}^{\frac{N}{4}-1} x(4m+2)W_N^{k(4m+2)} + \sum_{m=0}^{\frac{N}{4}-1} x(4m+3)W_N^{k(4m+3)}$$

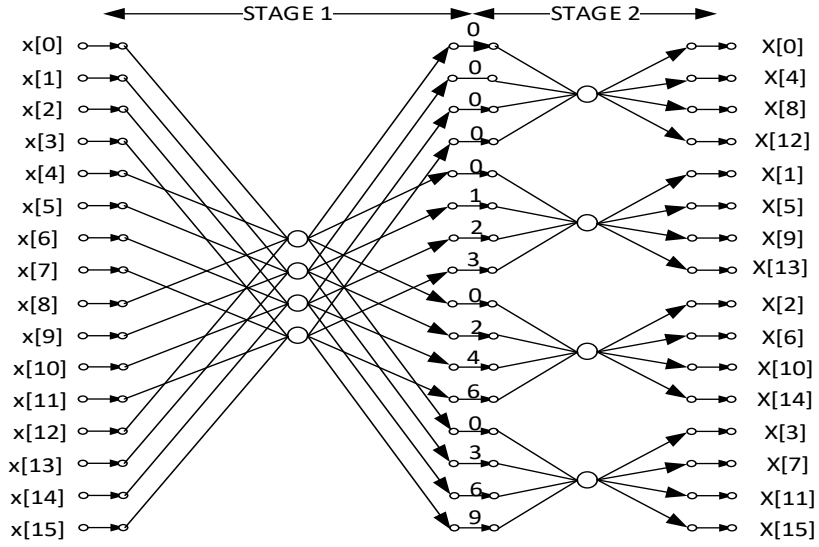

Figure 3. Signal flow graph of 16 point radix-4 DIF-FFT

4. RADIX- 2^i FFT ALGORITHM

4.1. Radix-2^2 algorithm

Radix-2^2approach proposed by He and Torkelson. By using linear mapping techniques, the two butterfly units are computed to one butterfly unit in radix-22 [11]. For N=16, radix-22 is computed in two stages but with different twiddle factorswhen compared to radix-2 algorithm. Figure 4 shows signal flow graph of 16 Point radix-2^2 DIF-FFT. Figure 5 shows signal flow graph of 16 Point radix-2^3 DIT-FFT.

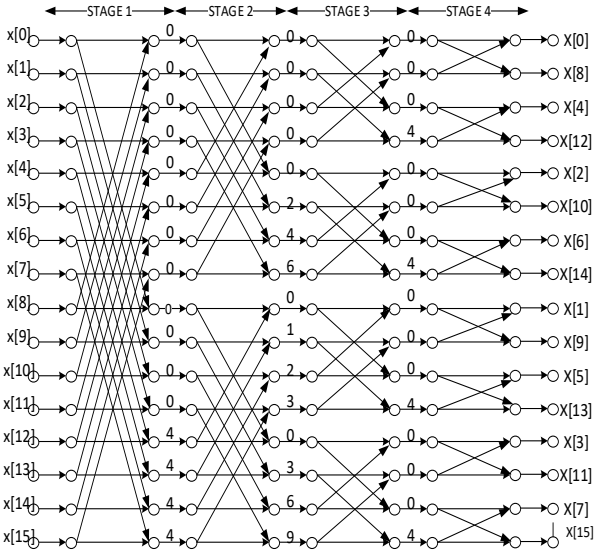

Figure 4. Signal flow graph of 16 point radix-2^2 DIF-FFT

The radix-2^2 is derived by writing the equation

$$\boldsymbol{X(k_1+2k_2+4k_3)} = \sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\sum_{n_1=0}^{1} \boldsymbol{x(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)W_N^{(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)}\, X(k_1+2k_2+4k_3)}$$
$$= \sum_{n_3=0}^{\frac{N}{4}-1}\left[\boldsymbol{H(k_1,k_2,n_3)W_N^{n_3*(k_1+2k_2)}}\right] * \boldsymbol{W_{\frac{N}{4}}^{n_3*k_3}}$$

Where $\boldsymbol{0 \le n_3 \le \frac{N}{4}-1,\ 0 \le k_3 \le \frac{N}{4}-1}$

$$\boldsymbol{H(k_1,k_2,n_3)} = \left[\boldsymbol{x(n_3)+(-1)^{k_1}*x\left(n_3+\frac{N}{2}\right)}\right] + \boldsymbol{(-j)^{(k_1+2k_2)}} * [\boldsymbol{x\left(n_3+\frac{N}{4}\right)+(-1)^{k_1}*x\left(n_3+\frac{3}{4}N\right)}]$$

4.2. RADIX-2^3 ALGORITHM

From the figure, it indicates that in radix-2^3 algorithm the twiddle factor exists in third stage [12]. The equation of radix-2^3 algorithm can be derived as follows

$$\boldsymbol{X(k_1+2k_2+4k_3+8k_4)} = \sum_{n_4=0}^{\frac{N}{8}-1} [\boldsymbol{BU_{\frac{N}{8}}^{k_1k_2k_3}(n_4)W_N^{n_4(k_1+2k_2+4k_3)}W_N^{n_4 8k_4}}]$$

Where

$$\boldsymbol{BU_{\frac{N}{8}}^{k_1k_2k_3}(n_4)} = \boldsymbol{BU_{\frac{N}{4}}^{k_1k_2}(n_4)} + \boldsymbol{BU_{\frac{N}{4}}^{k_1k_2}\left(\frac{N}{8}+n_4\right)}\boldsymbol{[0.707(1-j)]^{k_1}.(-j)^{k_2}(-1)^{k_3}}$$

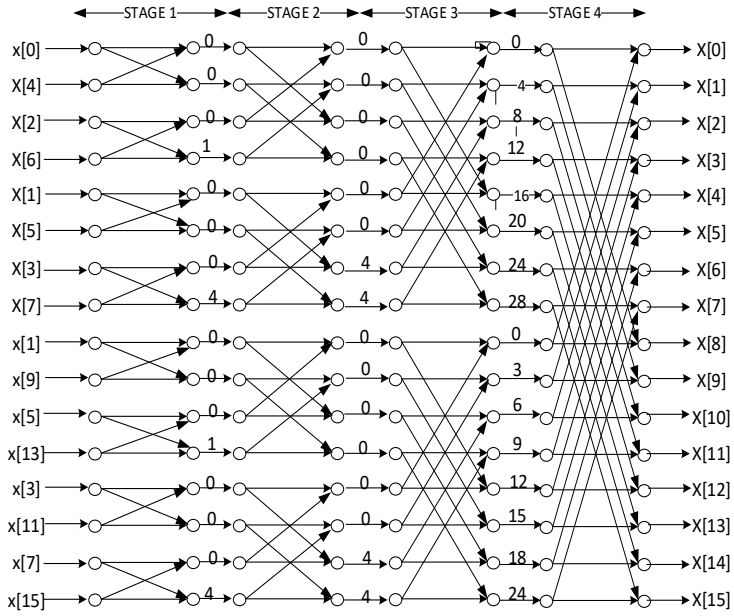

Figure 5. Signal flow graph of 16 point radix-2^3 DIT-FFT

5. PROPOSED WORK

In this paper, the proposed work demonstrates using the different combinations of the computational elements and implementation of those computational elements (multipliers and adders) in different radices such as radix-2, radix-4, radix-2^2 and radix-2^3 FFT architectures for 16-point. The overall impact and performance is considered in different radices using different computational elements. The work focuses on the FFT architecture and the computations to be done in each butterfly unit in the radix using the different combinations of multipliers. From the above signal flow graphs, it is illustrated that to compute twiddle factor with the input, the multiplication is necessary. So efficient multiplier should be considered to have an efficient radix in FFT processors.

Firstly, different computational elements (multipliers and adders) have been studied and utilized in different radices to acquire the efficient FFT architecture. In this paper, the different multipliers used such as Booth multiplier, Modified Booth [13], Canonical signed Digit (CSD) [6], multipliers to compute twiddle factor in butterfly unit. The adder used is carry save adder as it is faster and more efficient when compared to carry-look ahead adder.

5.1. Twiddle Factor Multiplication

Twiddle factor multiplication plays significant role in solving the butterfly unit in each stage of the different radices. While multiplying the twiddle factor with the value, an efficient multiplier is used in the radix. Different twiddle factors used in 16-point radices are represented as:

$$\boldsymbol{W_N^0, W_N^1, W_N^2, W_N^3, W_N^4, W_N^5, W_N^6, W_N^7, W_N^8, W_N^9}.$$

From the above figure, it is observed that how the multiplication and addition process occur in the butterfly units in the radix. Wherever the addition requires adder is used in that place and for multiplication different above mention multipliers can used. Twiddle factor values are represented as 0.707, 0.923.0.382, these values are converted into binary form and then represented in Q-format (fixed point representation) [14]. The twiddle factor values represented in Q-format are shown below in the Table 1. Figure 6 shows diagram of butterfly unit.

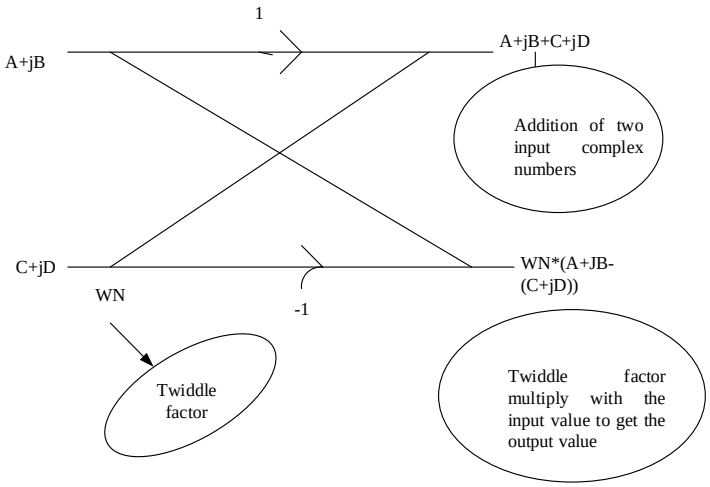

Figure 6. Diagram of butterfly unit

Table 1. Twiddle factor values represented in radix-2 DIT FFT

TF value	Binary value	Q-format value
0.707	+(1011_0100_1111_111)	0_00000_1011010011
-0.707	-(1011_0100_1111_111)	1_00000_1011010011
0.923	+(1110_1100_0100_101)	0_00000_1110110001
-0.923	-(1110_1100_0100_101)	1_00000_1110110001
0.382	+(0110_0001_1100_101)	0_00000_0110000111
-0.382	-(0110_0001_1100_101)	1_00000_0110000111

5.2. Fixed point Multiplication

Fixed point multiplication has different method to multiply two signed or unsigned numbers. Fixed point number representation the number of digits or bits are fixed either before or after the radix point i.e. binary point. Figure 7 shows format of fixed-point number.

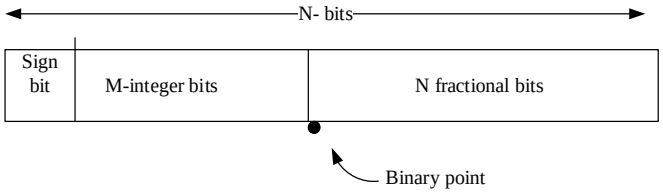

Figure 7. Format of fixed-point number

6. RESULTS

This paper deliberates the implementation of 16- point radices using the different combination of computational elements in the butterfly unit of FFT architecture. The analysis and comparison are made between different radices of FFT. The functionality of different multiplier modules was verified by running the test benches, simulations and synthesis in Xilinx ISE 14.7 tools using Spartan 6 family. From the above Tables 2 and 3, it is illustrated that the Vedic multiplier consumes less LUTs (502) and canonical signed digit consumes less delay 18.052ns for 16-bits. Carry save adder is faster when compared to carry look ahead adder.

From the above Tables 4 and 5, it is observed that

Comparison result of different radices (radix-2, radix-2^2 and radix-2^3) in percentage:

a) Radix-2^3 consumes 43% less LUTs when compared to the radix-2.
b) Radix-2^2 increases the frequency(MHZ) about to 40% when compared to radix-2.
c) Radix-2^3 consumes 17% less power(W) when compared to radix-2.

Comparison result of different radices (radix-4 and radix- 2^2) in percentage:

a) Radix-2^2 consumes 26% less LUTs than radix-4.
b) Radix-2^2 increases the frequency (MHZ) about to 19% when compared to radix-4.
c) Radix-2^2 consumes 26% less power(W) when compared to radix-4.

Table 2. Comparison between different multipliers

Types of multiplier	Number of bits	Slice LUTs	Delay(ns)
Modified Booth	8bit	202	18.295
	16bit	824	31.526
Canonical signed digit (CSD)	8bit	203	13.25
	16bit	916	18.052
Vedic	8bit	106	18.176
	16bit	502	18.399

Table 3. Comparison between different adders

Types of adders	Number of bits	Slice LUTs	Delay(ns)
Carry-Save Adder using RCA at last stage	8 bits	22	10.855
	16 bits	45	17.344
Carry Look Ahead Adder (CLA)	8 bits	14	8.841
	16bits	28	12.433

Table 4. Comparison between radix-4 and radix-2^ 2

Computational Elements		Radix-4			Radix-2^2		
Multiplier	Adder	Slice LUTs	Freq (MHZ)	Power (W)	Slice LUTs	Freq (MHZ)	Power (W)
Modified Booth	CSA	3227	61.126	0.210	3171	62.187	0.155
CSD	CSA	4316	53.154	0.155	4433	53.106	0.115
Vedic	CLA	4153	51.298	0.199	3669	50.232	0.170

Table 5. Comparison between radix-2, radix-2^2 and 2^3 FFT algorithms

Computational Elements		Radix-2			Radix-2^2			Radix-2^3		
Multiplier	Adder	Slice LUTs	Freq (MHZ)	Power (W)	Slice LUTs	Freq (MHZ)	Power (W)	Slice LUTs	Freq (MHZ)	Power (W)
Modified Booth	CSA	3179	46.39	0.22	3171	62.187	0.155	2957	54.68	0.177
CSD	CSA	5251	37.278	0.177	4433	53.106	0.115	5101	45.297	0.218
Vedic	CLA	4024	42.053	0.253	3669	50.232	0.17	3641	50.414	0.208

7. CONCLUSION

This paper has presented an efficient multiplicative and additive method for the FFT algorithm, where various combinations of computational elements were discussed, placing the emphasis on the butterfly units of the different radices such as radix-2, radix-4, radix-2^2 and radix-2^3 FFT algorithms. These different radices algorithm has been realized by Verilog code. The synthesis results shown in the Tables 1 and 2 confirms the efficient radix by computing the butterfly units with different combinations of computational elements. In this paper, the architectures for 16-point FFTs are analyzed, and the architectures with different computational elements are simulated and synthesized. The synthesis result shows the best performance in terms of LUTs and frequency (MHZ). In summary, our proposed work is to reduce power consumption, reduces the area and increases the speed of FFT architectures.

REFERENCES

[1] M. Aaravind Kumar and Dr. k Manjunath Chari, "Complex-multiplier implementation for pipelined FFTs in FPGAs", *IEEE conference*, 2015.
[2] Behrooz parhami, "computer arithmetic: algorithms and hardware designs", oxford university press, inc. New york, ny, usa ©2009.
[3] Amir Kaivani, and Seok-bum Ko, "Area efficient floating-point FFT butterfly architectures based on multi-operand adders", *Electronics Letters*, vol.51 no.12, 2015.
[4] E. Wold and A. Despain, "Pipeline and parallel-pipeline FFT processors for VLSI implementations," *IEEE transactions on computers*, vol. c- 33, no. 5, pp. 414–426, 1984.

[5] Ma, Z.-G., Yin, X.-B., Yu, F.: " A novel memory-based FFT architecture for real-valued signals based on a radix-2 decimation-in-frequency algorithm*". IEEE trans*. circ. syst. ii: exp. briefs 62, 876–880 (2015).
[6] R. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE trans. circuits & syst. ii*, vol. 43, oct. 1996.
[7] Sidinei Ghissoni, Eduardo costa, Ricardo reis, "Reusing smaller optimized FFT blocks for the realization of larger power- efficient radix-2 FFTs", *IEEE conference*, 2015.
[8] Lakshmi Santhosh, Anoop Thomas, "Implementation of radix-2 and radix-2^2 FFT algorithms on spartan6 FPGA" *IEEE conference*, 2013.
[9] S.Bouguezel, M.O.Ahmad, M.N.S. Swamy, "A new radix-2/8 FFT algorithm for length- qx2^m DFTs", *IEEE transaction*, volume no 51, 2004**.**
[10] Amirfattahi, "Calculation of computational complexity for radix-2p fast fourier transform algorithms for medical signals", *Journal of Medical Signals and Sensors*, vol 3, no 4, 2013.
[11] J. Cooley, P. Lewis, and P. Welch, "Historical notes on the fast fourier transform," *proc. IEEE,*vol. 55, no. 10, pp. 1675–1677, oct. 1967.
[12] M. Shin and H. Lee, "A high-speed four-parallel radix-24 FFT processor for UWB applications," *in proc. IEEE int. symp. circuits syst*. (ISCAS), 2008, pp.960–963.
[13] A. Cortés, I. Vélez, and J. F. Sevillano, "Radix rk FFTs: matricial rep- representation and sdc/sdf pipeline implementation," *IEEE trans. signal process*., vol. 57, no. 7, pp. 2824–2839, jul. 2009.
[14] Arish. S and R.K. Sharma, "Run-time configurable multi-precision floating point multiplier design for high speed, low power applications", *IEEE conference on spin*, 2015.